PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 3/023, 17/27

A1

(11) International Publication Number: WO 00/38041

(43) International Publication Date: 29 June 2000 (29.06.00)

(21) International Application Number: PCT/GB99/04321

(22) International Filing Date: 20 December 1999 (20.12.99)

(30) Priority Data:
9827930.0 19 December 1998 (19.12.98) GB

(71) Applicant *(for all designated States except US)*: SYMBIAN LIMITED [GB/GB]; Sentinel House, 16 Harcourt Street, London W1H 1DS (GB).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: HEALEY, Nicholas [GB/GB]; 4 Freeland Road, London W5 3HR (GB).

(74) Agent: ORIGIN LIMITED; 24 Kings Avenue, London N10 1PB (GB).

(81) Designated States: CN, GB, JP, US, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: KEYBOARD SYSTEM FOR A COMPUTING DEVICE WITH CORRECTION OF KEY BASED INPUT ERRORS

(57) Abstract

A method of correcting a key based input to a computing device using keys of a keyboard, including the step of determining the keys adjacent to the keys actually struck or selected. Hence, if the word 'car' is accidentally input as 'xar', the present invention includes the step of determining that the letter 'x' lay adjacent to the letters 'c' and 'z', assuming that a conventional QWERTY keyboard had been used. This information can be used to automatically correct the mis–hit.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

Keyboard system for a computing device with correction of key based input errors

Field of the Invention

This invention relates to keyboard systems, and in particular to keyboard systems for computing devices which can detect and correct key-based input errors. The term 'key' used in this patent specification should be expansively construed to cover any form of input system which a user touches, presses or otherwise selects in order to identify a letter, number or function. It covers, without limitation, the numeric and alphameric keys of a mechanical keyboard, a soft key array and a touch panel. A 'keyboard' as used in this specification is a collection of such 'keys'. The term 'computing device' used in this patent specification should be expansively construed to cover any form of electrical device and includes computers of any form factor, including handheld and personal computers, and communication devices of any form factor, including mobile telephones, smart phones, communicators which combine communications and computing functionality into a single device and other kinds of wireless and wired information devices.

Description of the Prior Art

Sophisticated word processing programs such as Microsoft Word include advanced spell check programs which can compare words input to a computer using a conventional QWERTY keyboard against words in a dictionary stored in the computer. Spell check programs are also found in hand held, self contained units, such as those from Franklin Computer Corporation. Spell-check programs have been the subject of intense scrutiny for several decades and many different approaches have been devised for effective spell-checking. Reference, for example, may be made to US 5248536 in the name of Franklin Electronic Publishers, Inc. which discloses a spell-check program which includes comparing an input word to a dictionary using phonetic comparison, typographic comparison, vowel and consonant typographic comparison and consonant phonetic comparison. The exact form of comparison which is undertaken in contemporary spell check programs involves sophisticated pattern matching algorithms which are disclosed in the literature.

One of the limitations of current spell-check programs can be best illustrated by an example. Suppose the word 'car' is to be input using an ordinary QWERTY keyboard. If the letter 'c' is accidentally missed, and instead the adjacent letter 'x' is hit, many conventional spell check program will in essence present the user with a list of all three letter dictionary words ending in

'ar'. Hence, the following list might be presented to the user in a list of possible 'correct' words:

ear oar bar car far gar jar mar par tar war

This requires the user to reject the first three candidate 'correct' words before selecting the genuinely correct substitute, 'car'. This is time consuming and can be tedious for the user.

For some three letter combinations, an even more restrictive list will be generated. For example, if the word 'cat' is erroneously input as 'xat', then many conventional word-processing spell-check programs will suggest only 'at' as the correct substitution.

Hence, prior art spell-check programs do not consider whether or not a key that has been hit or selected is actually adjacent to the key that should have been struck.

The problem of accidentally hitting a key adjacent to the correct key affects users of full sized QWERTY keyboards, but may be particularly acute where miniature keyboards are used.

Touch screen keyboards, such as are used in personal organisers and some communicators, are particularly prone to mis-hits because of the small size of the keyboard and the absence of discrete keys. Reduced keyboards, for example incorporating nine keys, each being labelled with three letters, are one solution to the problems posed by text entry to small hand held devices. Reference may for example be made to US 5818437 to Tegic Communications, Inc. Reduced keyboard systems, however, do not compensate for accidentally hitting a key adjacent to the correct key.

Statement of the Invention

In accordance with a first aspect of the present invention, a method of correcting a key based input to a computing device using keys of a keyboard includes the step of determining the keys adjacent to the keys actually struck or selected. The key based input may relate to letters.

Hence, returning to the example given above of the word 'car' accidentally being input as 'xar', the method of the present invention would include the step of determining that the letter 'x' lay adjacent to the letters 'c' and 'z', assuming that a conventional QWERTY keyboard had been used. That determination can be used in a variety of ways. For example, it could be used to weight the results obtained through conventional pattern matching spell check

techniques, so that the word 'car' was presented as the first, or in some embodiments, the only 'correct' substitution. Prior art systems do not compensate for key mis-hits in this manner, or at all.

The key input may also not relate to letters at all. Keyboards on small computing or communications devices are very crowded and mis-selection of menu items is easy: for example, the on-screen menu on a primarily numeric keypad might dictate hitting any of the numbers '1', '2' or '3' to select one of three different options. If the key for number '4' is accidentally mis-hit instead of '1', then in one embodiment, the step of determining the keys adjacent to the key actually struck occurs. This enables the embodiment to determine that in all likelihood the user intended to strike key '1'.

In another embodiment, sophisticated pattern matching is dispensed with entirely; instead, there is a simple dictionary or N-gram look up based solely on words or letter strings generated using the various letter combinations associated with the keys either side and on the same row of the hit/selected keys, as well as the actually hit/selected keys themselves. Hence, for example, if the user was attempting to type the word 'soak', the letters 'doak' might be hit. Then, the in the present method, the following letter combinations are considered in the dictionary look-up (again, assuming a QWERTY keyboard):

s	d	f
i	o	p
	a	s
j	k	l

One word in the dictionary that can be extracted from these possible combinations is 'soak'. This approach is quite different to the prior art approach, in so far as it models the errors in input words that can arise through mis-hits.

Optionally, the present invention envisages a method in which, if there is more than one word located in the dictionary process (whether or not that process uses any of the sophisticated pattern matching techniques found in conventional spell-check programs), then there is a further step of determining which word located in the dictionary look-up process includes the highest number of letters actually hit. It is that word that may be made the highest priority suggested or automatic substitution. Hence, in the previous example, both 'foal' and 'soak' and 'dial' are possible words. The ambiguity relating to which of these 3 words was intended

can be resolved by determining which of these words includes the highest number of letters actually hit. In this example, the word 'soak' would be correctly identified, since it has 3 letters actually hit compared to 'foal' and 'dial', which each have 2 letters.

In the above embodiments, one of the inventive steps is that there is a determination of the keys on the same row which are adjacent to the keys actually struck or selected. In another embodiment, this principle is extended to cover a determination of keys which are merely nearby to the keys actually struck, such as keys which are next door but one on the same row, and keys which are on different rows but are nevertheless adjacent. Weighting of the significance of hitting such keys may also be used, with keys adjacent and on the same row being given the highest weighting.

For touchscreen keyboard embodiments of the invention, the keyboard may be extended by the inclusion of extra blank keys at each end of each row. Hence, for example, if the blank key adjacent to the key 'Q' on a QWERTY keyboard is hit, then the embodiment determines that the correct key should be the key 'Q'.

In addition, monitoring of the exact touch position may also occur. This enables weighting of the proximate letters closest to the actual contact position. Hence, on a typical touchscreen, contact positions for any one key may be resolved using an 8x8 grid touch sensitive grid overlying the display, with all keys being defined by their own 8x8 grid. Hence, if the touched gird defines the letter 'H' on a QWERTY keyboard, but the actual contact position is clearly closer to 'G' than 'J' on the 8x8 grid, then the letter 'G' can be weighted more heavily in the word selection process.

For ordinary mechanical keyboards (i.e. those relying on a degree of travel of a key to activate selection of that key), ambiguity can arise when a finger accidentally presses 2 keys at the same time. That risk is increased as keyboard size decreases. This kind of mis-hit normally results in both letters being selected: hence if on a QWERTY keyboard, an attempt is made to spell 'cat' but both 'c' and 'v' keys are simultaneously hit, then the word 'cvat' or perhaps 'vcat' would ordinarily be input. In one embodiment, pairs of letters which are adjacent to one another are recognised and, if the input word is not recognised as a dictionary word, then the system selects one of the adjacent letters forming a pair and performs a dictionary look up. Hence, the system might select the 'c' in 'cvat' and perform the look-up against 'cat'. Likewise, the system could select the 'v' in 'cvat' and perform the look-up against 'vat'.

In another embodiment, the system of the present invention may be adapted so that dictionary words which are very rarely used are subject to the letter substitution techniques described above. Hence, the word 'See' may be incorrectly input as the word 'Dee' because of a mis-hit of the letter 'D' instead of 'S'. A conventional spell-check program might pass this if it contained the unusual word 'Dee' in its dictionary (it is the name of a British river). In this embodiment, the word 'Dee' is classified as an unusual word, so that the embodiment automatically applies letter substitution to generate the more common word 'See'. The more common word 'See' is then offered as a potential substitution, or may be automatically input.

It may also be desirable to dynamically modify the 'probable mis-hit' table in the light of experience. For example, some users may be more prone to hitting keys on the same row, others on the same column. Further, the mis-hitting may vary from one part of the keyboard to another.

In addition, as well as or instead of using a dictionary, other data structures such as an n-gram approach could be used to validate words. N-grams would require less memory, and be an intrinsically faster algorithm than a dictionary based approach.

Feedback to the user can also be provided, indicating the proposed word correction. This enables the user to correct the proposed correction in an efficient manner, and also to modify the data structures used in the word validation process. In this way, the process can be an adaptive one.

In other aspects, there are provided: a computing device operable to perform the above inventive methods; pre-recorded media programmed with software to perform the above inventive methods; and a computer program operable to perform the above inventive methods.

Detailed Description

The following Appendix 1 is presented as the Detailed Description.

Appendix 1

This section describes a number of algorithms for implementing the present invention. Each is documented in 'pseudo-code', which should be translatable into most programming languages by a competent programmer.

Algorithm One: High Level

This algorithm uses the nearest match routine that is to be found with many dictionary systems.

```
word$ = Obtain word from keyboard input
Look up word$ in dictionary
If not found in dictionary then
     Propose nearest match
```

Algorithm Two: NearestMatch

This is a complete embodiment in that it can be compiled and run and was tested using Borland Delphi v 5 (Pascal) on a PC running Microsoft Windows '95. However, not all techniques included in the patent body are necessarily illustrated in the source code. All techniques (other than those specifically identified within the source code) are part of the patent. The Pascal language was used since it was designed for illustrating algorithms. This patent may be embodied in any computer language or indeed in hardware. First we present the source code which teaches an embodiment of the present invention. Any line numbers are part of this patent documentation, rather than part of the code. Formatting is for guidance only. Comments are enclosed within curly brackets, {like this}.

```
Program Keypad;

{$APPTYPE CONSOLE}

uses
      SysUtils;

{  ====  Dictionary  ====  }

{  The dictionary implementation presented here is not part of the
   invention, but is used to show how the invention might
   interact with a dictionary or any other structure capable of verifying
   the correctness of a word. }

{  This demonstration dictionary, whose implementation is not relevant,
   is implemented as a simple binary tree, a well recognised technique
   in computer science. }

Type
   DictionaryPointer =  ^DictionaryNode;
   DictionaryNode =  Record
      less,                          {  Pointer to lesser sub-nodes   }
      more: DictionaryPointer;       {  Pointer to greater sub-nodes  }
      text: String;                  {  The word being stored         }
      {  info: data; }               {  Extra information here        }
   End;

Var
   Dictionary: DictionaryPointer;    {     The acutal dictionary      }

Function LookUp(  Const d: DictionaryPointer;
                  Const s: String): Boolean;
   {  Returns true if s is found in the dictionary d.
      The actual implementation is not part of the patent.  }
   Begin
      If d = Nil then
         LookUp:= False
      Else with d^ do begin
         If text = s then
            LookUp:= True
         Else if s < text then
            LookUp:= LookUp(less, s)
         Else
            LookUp:= LookUp(more, s)
      End;
   End;

Procedure Insert( Var   d: DictionaryPointer;
                  Const s: String);
   {  Inserts s into the dictionary }
   Begin
   {  Is this an empty node?  }
      If d = Nil then begin
      {  Yes, so put in the word }
         New(d);
         With d^ do begin
```

```
            less:= Nil;
            more:= Nil;
            text:= s;
         End;
      End
      Else with d^ do begin
      { No, its not an empty node. }
         If text = s then
      { A duplicate - but this implementation does not mind      }
            Exit
         Else If s < text then
         { It is less then the word stored here,
            so insert it in the lesser sub-tree.      }
            Insert(less, s)
         Else
         { It is greater than the word stored here,
            so insert in in the greater sub-tree.     }
            Insert(more, s)
      End;
   End;

Procedure DisposeDictionary(Var d: DictionaryPointer);
   { Disposes of the dictionary with a post-order traversal      }
   Begin
      If d <> nil then begin
         With d^ do begin
         { Dispose of the lesser sub-tree      }
            DisposeDictionary(less);
         { Dispose of the greater sub-tree     }
            DisposeDictionary(more);
         { Clear out our data   }
            text:= '';
         End;
      { Dispose of this node }
         Dispose(d);
         d:= Nil;
      End;
   End;

{ ---- The following routines form the 'API' to our dictionary ---- }

Function InDictionary(Const s: String): Boolean;
   { Returns true if the word s is in the dictionary }
   Begin
      InDictionary:= LookUp(Dictionary, UpperCase(s))
   End;

Procedure LoadDictionary(Const FileName: String);
   { Loads the dictionary from a file of words.
      Each word must be one per line, with no spaces.
      The words should not be in alphabetical order.
      Only call this once! }
   Var
      f: TextFile;
      DictionaryWord: String;
   Begin
      Write('Loading ', FileName, ' ...');
```

```
      Dictionary:= Nil;

      AssignFile(f, FileName);
      ReSet(f);

      While not EoF(f) do begin
         ReadLn(f, DictionaryWord);
         Insert(Dictionary, UpperCase(DictionaryWord));
      End{while};

      CloseFile(f);

      WriteLn('done');
   End;

Procedure ClearDictionary;
   { Clears out the current dictionary   }
   Begin
      If Dictionary <> Nil then
         DisposeDictionary(Dictionary)
   End;


{  ====  Core Algorithm ====  }

{  The code in this section is part of the patent.
   This embodiment is a very restricted one, designed to show the
   principles.
   For example, it only handles the letters on a standard keybaord.     }

{  Tables   }

{  This table of neighbouring keys, within the scope of the patent,
   could be made:
   1) To hold further information, such as probabilities and scores
   2) Adaptive, to learn which keys the particular user tends to press in
      error, as well as adapting to different parts of the keyboard.    }

Var
   Neighbours  :  Array['A'..'Z'] of String =
      (  {a}   'SQZW',        {b}   'VNGH',
         {c}   'XVDF',        {d}   'SFEXC',
         {e}   'WRD',         {f}   'DGRCV',
         {g}   'FHTVB',       {h}   'GJYBN',
         {i}   'UOK',         {j}   'HKUNM',
         {k}   'JLIM',        {l}   'KO',
         {m}   'NJK',         {n}   'BMHJ',
         {o}   'IPL',         {p}   'O',
         {q}   'WA',          {r}   'ETF',
         {s}   'ADWZX',       {t}   'RYG',
         {u}   'YIJ',         {v}   'CBFG',
         {w}   'QES',         {x}   'ZCSD',
         {y}   'TUH',         {z}   'XAS' );


Function OneKeyErrorCheck(Const InputWord: String): String;
   {  Using the Symbian technology,
```

```
      corrects a string for incorrect input.    }

   {  It takes the entered word, and returns a list of possible 'correct'
      words }

   {  The algorithm is that of subsituting each neighbouring key for each
      letter, to find all related varients in the dictionary.    }
   Var
      testWord ,
      possibles: String;
      letter   : Char;
      i, j     : Integer;
   Begin
      WriteLn('OneKeyErrorCheck:');

   {  Now see if we can create a valid word by transposing a single
      character to any of its keyboard neighbours.    }

      possibles:= '';

   {  Check each character position in turn...  }
      For i:= 1 to Length(InputWord) do begin
         testWord:= UpperCase(InputWord);
         letter:= InputWord[i];
      {  ... against each of its keyboard neighbours  }
         For j:= 1 to Length(Neighbours[letter]) do begin
            testWord[i]:= Neighbours[letter][j];
         {  If it is in our dictionary... }
            If InDictionary(TestWord) then begin
            {  ... then this word is possible, so add it to the list    }
               possibles:= possibles + ' ' + testWord;
               Write(testWord, '-ok ');
            End{then}
            Else
            {  ... otherwise it is not    }
               Write(testWord, '-no ');
         End{for j};
         WriteLn;
      End{for i};
      OneKeyErrorCheck:= possibles;
   End{OneKeyErrorCheck};

Function MultipleKeyErrorCheck(Const inputWord: String): String;
   {  This function assumes that more than one key may have been mistyped.
      It tries to find words with multiple key substitutions that are in
      the dictionary.

      This routine also illustrates the option of determining the most
      likely word as being the one with the minimum transformations.

      This routine could also handle the one-key error case.    }

   Function TryWordTransform( Const position    : Integer;
                                    changes     : Integer;
                                    s                 : String): String;
      {  By transforming letters at position, and calling itself,
         TryWordTransform returns all words it can generate which are in
         the dictionary.
```

```
        This algorithm is recursive because we do not know in advance how
        long the word is going to be.

        Non-recursive embodiments are also possible and part of the
        patent.

        As a variation, we are also calculating the number of changes
        (via the changes variable) required
        to get from the input word to the word in the dictionary,
        and using this as a score.  Note that other scoring systems, or
        indeed no scoring system, are also within the scope of this
        patent.  }
      Var
        letter    : Char;
        i         : Integer;
        possibles: String;
      Begin
        If position > Length(s) then
        {  We've got to the end of the word     }
          TryWordTransform:= ''
        Else begin
        {  Generate, test, and add to the list any valid varients
          which don't involve changing the current character.    }
          possibles:= TryWordTransform(position+1, changes, s);

        {  We now change the current character to all possibilities based
          on our table of neighbours.    }
          letter:= s[position];
          For i:= 1 to Length(Neighbours[letter]) do begin
            s[position]:= Neighbours[letter][i];
          {  See if this swap has generated a valid word...      }
            If InDictionary(s) then begin
            {  ... it has, so add it to the list    }
              Write(s, '-ok ');
              possibles:= Possibles + ' ' + s + '(' +
                                      IntToStr(changes+1) + ')';
            End
            Else
            {  ... it has not, so ignore it  }
              Write(s, '-no ');
          {  Generate, test, and add to the list any valid varients
            which derive from this changed version.       }
            possibles:= possibles + TryWordTransform(position+1,
                                                changes+1, s);
          End;
          TryWordTransform:= possibles;
        End;
      End;

    Begin
      WriteLn('MultipleKeyErrorCheck:');
      MultipleKeyErrorCheck:= TryWordTransform(1, 0, inputWord);
      WriteLn;
    End;

Function KeyTransformErrorCheck(Const s: String): String;
  {  Checks for key swaps.    }
```

```
    { This is not part of the Symbian invention,
      but is a well-known technique which might be used alongside it.   }
    Var
      testWord ,
      possibles: String;
      i        : Integer;
      c        : Char;
    Begin
      WriteLn('KeyTransformErrorCheck:');

      possibles:= '';

      For i:= 1 to Length(s)-1 do begin
      { Build a new test word...   }
        testWord:= s;
      { ...by swapping letters. }
        c:= testWord[i];
        testWord[i]:= testWord[i+1];
        testWord[i+1]:= c;
      { If its in the dictionary...      }
        If InDictionary(testWord) then begin
        { ... then it is a possibility  }
          Write(testWord, '-ok ');
          possibles:= Possibles + ' ' + testWord;
        End
        Else
        { ... otherwise it is not }
          Write(testWord, '-no ');
      End;

      WriteLn;
      KeyTransformErrorCheck:= possibles;
    End;

Function CheckWord(s: String): String;
    { This wrapper function calls a variety of techniques,
      all covered by the Symbian invention,
      to determine what the correct word might be.

      Note that the patent envisages other related techniques
      in addition to those documented in this simple embodiment.  }
    Var
      possibleWords: String;
    Begin
      s:= UpperCase(s);

      If InDictionary(s) then begin
        CheckWord:= s;
        Exit;
      End;

      possibleWords:= OneKeyErrorCheck(s);
      If possibleWords <> '' then begin
        CheckWord:= possibleWords;
        Exit;
      End;

      possibleWords:= MultipleKeyErrorCheck(s);
```

```
      If possibleWords <> '' then begin
         CheckWord:= possibleWords;
         Exit;
      End;

      possibleWords:= KeyTransformErrorCheck(s);
      If possibleWords <> '' then begin
         CheckWord:= possibleWords;
         Exit;
      End;

      CheckWord:= s + '?';
   End;

{  ====  Driver demonstration program      ====  }

Var
   TestWord: String;

begin{Keypad}
   WriteLn('Symbian Keyboard Patent Demonstration');
   WriteLn('Algorithum subject to Symbian patent');
   WriteLn;

   LoadDictionary('words.txt');

   Repeat
      Write('Enter test word, including ''mistakes'',);
      WriteLn('or just press <enter> to finish.');
      Write('>');
      ReadLn(TestWord);
      If Length(TestWord) > 1 then
         WriteLn('-> ', CheckWord(TestWord));
   Until TestWord = '';
   WriteLn;

   ClearDictionary;

   WriteLn('Finished');
end.
```

We now present an extremely simple test dictionary, refered to as 'words.txt' in the source code:

```
cat
rat
mat
sat
hat
bat
vat
fat
```

Claims

1. A method of correcting a key based input to a computing device using keys of a keyboard, including the step of determining the keys adjacent to the keys actually struck or selected.

2. The method of Claim 1 wherein a key based input represents a letter of the alphabet.

3. The method of Claim 1 wherein a key based input represents the selection of an option.

4. The method claimed in Claim 2 wherein the result of the determination is used to weight spell checking results obtained through conventional pattern matching techniques.

5. The method of Claim 2 including the further step of a dictionary or N-gram validation process based solely on words or letter strings generated by considering the various letter combinations associated with the keys either side and on the same row of the hit/selected keys, as well as the actually hit/selected keys themselves.

6. The method of Claim 5 wherein if there is more than one word located in the dictionary or N-gram validation process, then there is a further step of determining which word or string located in the dictionary or N-gram validation process includes the highest number of letters actually hit.

7. The method of Claim 2 and any preceding Claim dependent on Claim 2 wherein there is provided a blank key at the end of at least one row.

8. The method of Claim 2 and any preceding Claim dependent on Claim 2 further including the step of recognising pairs of letters which are adjacent on the keyboard and determining any words located in a word validation process which include one or the other of the letters forming the pair.

9. The method of any preceding Claim including the further step of the automatic correction of mis-hits.

10. The method of any preceding Claims 1-8 including the further step of the user being presented with one or more options, each corresponding to a possible key input or series of key inputs which the system considers to be a valid key input or series of key inputs.

11. The method of any preceding claim further including the step of determining the keys adjacent to the keys actually struck or selected but which are not immediately proximate.

12. The method of Claim 11 wherein the determination of keys which are adjacent but not immediately proximate to the keys actually struck is used to weight a spell checking or dictionary look-up set of results.

13. The method of any preceding claim including the step of monitoring the exact contact point for a key within the boundaries of the key itself.

14. The method of Claim 13 wherein the result of monitoring the exact contact point for a key within the boundaries of the key itself is used to weight a spell checking or dictionary look-up set of results.

15. The method of any preceding claim including the step of monitoring the time a key is selected and if two letters keys are selected at substantially the same time, the further step of determining if the two keys are adjacent or proximate to one another and, if so, choosing only one of those letters to use in a verification step.

16. A computing device operable to perform the method defined in Claims 1 - 15.

17. Pre-recorded media programmed with software to perform the method defined in Claims 1 -15.

18. A computer program operable to perform the method defined in Claims 1 - 15.

INTERNATIONAL SEARCH REPORT

Inte .onal Application No
PCT/GB 99/04321

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F3/023 G06F17/27

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 748 512 A (VARGAS) 5 May 1998 (1998-05-05)	1,2,5,7, 9,13,14, 16-18
A	column 1, line 43 -column 2, line 42; claim 17; figures 2,4-6	4,6,8
X	PATENT ABSTRACTS OF JAPAN vol. 13, no. 294 (P-894), 7 July 1989 (1989-07-07) & JP 01 074612 A (MITSUBISHI ELECTRIC CORP.), 20 March 1989 (1989-03-20)	1,2,4-6, 9,11,12
A	abstract	8

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
1 March 2000	08/03/2000
Name and mailing address of the ISA	Authorized officer

INTERNATIONAL SEARCH REPORT

Inter ...nal Application No
PCT/GB 99/04321

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	PATENT ABSTRACTS OF JAPAN vol. 17, no. 348 (P-1566), 30 June 1993 (1993-06-30) & JP 05 046594 A (NEC CORP.), 26 February 1993 (1993-02-26)	1,2,5,8, 9,11
A	abstract	6
X	PATENT ABSTRACTS OF JAPAN vol. 16, no. 308 (P-1381), 7 July 1992 (1992-07-07) & JP 04 085660 A (MATSUSHITA ELECTRIC IND CO LTD), 18 March 1992 (1992-03-18)	1,2,4,5, 9
A	abstract	6,8,11
A	PATENT ABSTRACTS OF JAPAN vol. 18, no. 50 (P-1683), 26 January 1994 (1994-01-26) & JP 05 274076 A (NEC CORP), 22 October 1993 (1993-10-22) abstract	1,2,4-6, 8,9,16
A	PATENT ABSTRACTS OF JAPAN vol. 1995, no. 4, 31 May 1995 (1995-05-31) & JP 07 013666 A (MATSUSHITA ELECTRIC IND CO LTD), 17 January 1995 (1995-01-17) abstract	1,2,9,15

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inter. ...nal Application No
PCT/GB 99/04321

Patent document cited in search report		Publication date	Patent family member(s)	Publication date
US 5748512	A	05-05-1998	NONE	
JP 01074612	A	20-03-1989	NONE	
JP 05046594	A	26-02-1993	NONE	
JP 04085660	A	18-03-1992	NONE	
JP 05274076	A	22-10-1993	NONE	
JP 07013666	A	17-01-1995	NONE	